Fermi National Accelerator Laboratory

Summer Internship

Final Report

Predictive Model for Optimizing Grid Utilization

September 28,2017

Fermilab

Author:
Antonio Di Bello

Supervisor:
Michael Kirby

Contents

Abstract

One of the great challenges for utilizing distributed computing (Open Science Grid, Cloud computing, High Performance Computing, etc.) through a scheduling/queuing system is knowing when resources will become available for usage by an individual user or group. Our project consisted of using Deep Learning algorithms to model the response of the Fermilab General Purpose Grid (GPGrid) cluster and predict when a user request will be matched to available resources.

The algorithm uses the current conditions of GPGrid, submitting user priority, the resource request, and numerous other factors to estimate when jobs will start on the cluster. With the support of historical data from FIFEMon and Kibana we have created a model of response within the industry based Deep Learning frameworks TensorFlow. The model will assist users in understanding when their jobs will start, complete, and how to optimize their access to GPGrid.

As well, the successful development of predictive model has considerable application in cloud computing and would influence the decision engine utilized with HEPCloud, the next-generation infrastructure for High Energy Physics Cloud computing.

1 Introduction

Goal This project aims to offer at the final user an accurate estimation of the needed time so that the job he has submitted in the queue batch system could start its execution. In order to do that we have to collect the historical job data from the existing database, then develop a Deep Learning model that is able to make a prediction of the waiting time.

Scenario In the lasts years the demand for computation power has considerably increased. Especially in scientific research, all the major institutions have deployed a computing infrastructures used to run simulation and analysis of data experiment. Unlike in the past when the mainframe were highly used, nowadays all the users have to share the same high performance computing. But this effort were result insufficient to completely supply all the necessity, and to resolve this problem was built a very large scale distributed system that include center of calculus among all the world. In this situation, among many years, there are developed software to handle the common resources and to manage the access to them. In order to reach this goal were born many different policy that provide the most suitable behavior wanted respect to the environment. A very large range of algorithm were used to schedule the user requests in order to obtain a right allocation of resource and optimize their utilization. In a complex scenario like this, make a some sort of predict can be very hard because an eventually analytic model should consider too many parameters, sometimes also unpredictable. To fill this gap, a modern Deep Learning approach can help us to obtain not exact results, but useful ones. This is possible thanks to the huge number of data collected by the installed monitoring system. We first consider all the possible Deep Learning model, and once identify the most suitable, we train and tune that to obtain the smallest error possible over the test data. Finally we discuss the validity of our model, and at which point of trustworthiness could be used.

2 System Components

Fermilab has many grid computing facilities onsite, each of them used for specific purpose. The most important among them is the General Purpose Grid (GPGrid), that is used by all the current Intensity Frontier experiments. But GPGrid belong to a more large and complex reality. Indeed, together at many site situated worldwide, it is a part of the Open Science Grid, a large distributed computing infrastructure that provides resources for all the requests by the international scientific community.

2.1 Open Science Grid

The **Open Science Grid** (OSG) is a high-throughput distributed computing infrastructure designed to facilitate large-scale scientific research. Developed and operated by a consortium of universities, national laboratories, scientific collaborations, and software developers, the OSG interoperates with multiple grid infrastructures throughout the world, allowing scientists to have access to computing resources that they may not have been able to use otherwise. Internally, all the users are divided into Virtual Organizations.

A **Virtual Organization** (VO) is a set of groups or individuals defined by some common cyber-infrastructure need. This can be a scientific experiment, a university campus or a distributed research effort. A VO represents all its members and their common needs in a grid environment.

OSG brings together many VOs for contracted and/or opportunistic sharing of resources in a grid environment, allowing for more effective use of their collective resources. Use of resources at a site is determined by a combination of the site's local policies and the user VO's policies.

2.2 HTCondor

In order to optimize the managing of the system resource and guarantee a correct distribution of them among all the user, the Open Science Grid uses a batch system called HTCondor.

HTCondor is a specialized workload management system for compute-intensive jobs. Like other full-featured batch systems, HTCondor provides a job queueing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Users submit their serial or parallel jobs to HTCondor, HTCondor places them into a queue, chooses when and where to run the jobs based upon a policy, carefully monitors their progress, and ultimately informs the user upon completion.

Finally, to ensure an equity resources sharing, it is adopted a **fair schedule**. The goal of this schedule is to avoid that a few users could held the main part of resources, prevent the others users to access them. To do that, each job is placed inside the queue in a place determinate by the amount of requested resources (the user choose this parameter), his dynamic priority, the available resource inside the belonging VO, and other many parameters.

2.3 FIFEMon

All the system is constantly monitored by FIFEMon, a tool that constantly track all the parameters that could be analyzed at a later time. With this data, we have the possibility to study and model the system history at the right level of granularity, from the global aggregated job, passing by the VO information, until to the single job. Finally, the data are presented at the users trough **Graphana**, that render the data in a viewable mode such as plots and histograms.

3 Project Overview

In order to develop a valid Deep Learning model, we should first of all define which data could be useful to reach our purpose. This is an empirical choice because we don't know exactly the data required, and therefore if we believe that other data could improve our result, we can add them later.

Once focused the desired data, we proceed to **collect** them from the respective source and store in a appropriate format like CSV.

After that we can begin to **build** our model. We have to define the parameters and the structure, in particular the topology of the Neural Network.

Finally we have to **train** the model in order to make it learn our problem, and consequently **test** if it's doing it correctly. To do that, we randomly split the collected data into two set: one used to train and the other used to test. In this mode we can evaluate step by step our model and avoid to learn only this specific data (overfitting) and also to avoid insufficient learning from the data (underfitting).

When we obtain good results, we can optionally **deploy** the project into a preexisting environment, providing the GPGrid users with a tool to predict when their jobs will start processing and complete.

It's important to notice that in order to get better results we can optionally iterate between the precedent defined phases, changing some parameters or structure, adding other data, and so on.

4 Collecting Data

The features that we have identified as useful for modeling the system status and response can be divided into the ones related to the single jobs, and the ones related to the entire system.

4.1 Job Data

When a user submits a job, he creates a *cluster* composed by possible many *processes*. Furthermore the cluster is associated at one *batch* system that will handle it. Then ,we can refer at a single process using an ID that has the following structure: *cluster.process@batch*. All the process into one cluster share the same amount of resources, but obviously each of them has an independent execution flow.

An **execution flow** is characterized by many kind of events. A cluster born with a *Submit* event, that represent the event of the job submission by the user and its entry in the queue. After that, when the required resources are available and the job is the next scheduled for execution, we have an *execution* event. Finally when the job end its execution we record this with a *termination* event. Throughout the execution, a job could bump into a non-standard event, that temporally stops its computation. In order to retrieve only the relevant data, we discard all clusters that have at least one process that encountered a non-standard event.

We have highlighted the features related to jobs by listing the required resources that the user ask for the execution. These parameters are the number of **CPU**s, the amount of **memory** and **disk**, the estimated **time** needed to complete, the **site** to physically execute the computation and also the **experiment** (VO) of membership.

All this information are logged into a *Kibana* database, that we query using the *ElasticSearch* language.

4.2 System Data

We have a large quantity of parameters that describe the system. Making a decision about which of them are useful or not is very hard, so we have opted to collect only the total number of **running** and **idle** jobs. In addition in order to have information at a lower level of granularity, we have individualized a representative subset of the major experiments and for each of them, we retrieve the number of running and idle jobs and the **quota** of the experiment over the full system.

We retrieve this data from the *Graphite* database make a simple *GET* request.

4.3 Algorithm

Fist of all we collect the system data belonging to the defined interval of time. Trough the python *requests* library , we retrieve data with a delta time of 5 minutes for the lasts months, and of one hour for the further ones.

After that, we have to collect the job data from the database index *fifebatch-jobs.** that track every process event. Then we make an aggregation query that group first by batch and then by cluster. From this query we extract for each cluster, in order of submission time, the time of submission and the earliest time between all the execution start time of the process in the same cluster.

Now we have to filter this data, deleting all the cluster that have at least one process with a *bad* event. We called bad event every event that could interfere with the normal program flow. For that, we query *fifebatch-logs-**, another index of the same database. We don't retrieve any field, but we have interested only at the total number of match found. We discard all the cluster with non-zero matches.

Once we have only valid clusters, we retrieve all the information (like required resource) inherently the respectively job doing a GET request in which URL we put the ID cluster. In the last query we retrieve only the last process that terminate the execution, take the event time and use it as end time for the cluster.

Finally we calculate the wait time and delay time, put them together at the job and system data previously collected, and write the sample just built into a row in file.

5 Neural Network

Neural Networks are mathematical models that can efficiently learn from a huge number of examples, and apply this knowledge to resolve the same problem with data never seen. This is possible because internally a neural network is composed by neurons organized in layers connected between them with a weighted link that change its value while learning. Neural networks get good results in classification (for example of images) where we want to extract some high level features, or in time series prediction where it can learn from the past to predict the future. In our case we want a continuous output and not a classification, therefore we have a regression problem. We use the *TensorFlow* library to implement this model in our program.

5.1 Model

There are many kind of neural network, each of them specialized in some field. The most naive one is the **Linear** model, where we have one single layer and the output is a weighted sum of the input, plus a bias. This model is good for learning very specific rules but is weak when it have to generalize over cases never seen.

To obtain better results, we have to go towards more complicated models like **Deep** Neural Networks (DNN). In this network we have many **dense** layers called hidden layers, that are fully connected in series. Each layer is characterized by a number of neuron and an activation function, the function that generate the output. This model can learn better because it has more links, more variables, and tunable activation functions.

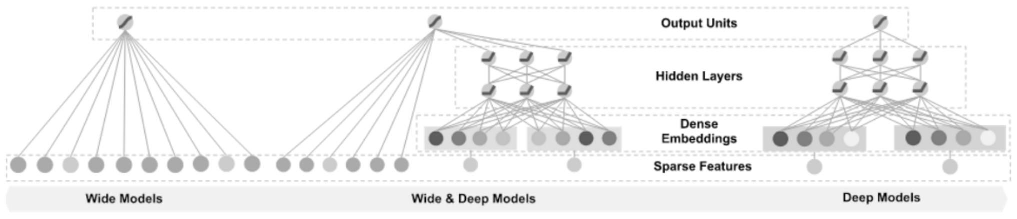

Figure 1: Representation of models in order of complexity from the linear to deep.

A last model that we believe that have to be mentioned is the **Convolutional** Neural Network (CNN). CNN is often used like extractor and classifier of complex features. This network is composed by an arbitrary number of alternate layer of convolution and pooling. In the **convolution** layer we expand the dimension space of input doing a convolution between it and the neurons of the layer that compose many kernels. Each kernel could extract some specific features, defined during the training. In the **pooling** layer, undergoes one of some aggregation operations like average or maximum. Here the dimension output is reduced in order to maintain only a general idea of the features, and its combination extracted. Finally there are an optionally dense layers and **dropout** layers. In the dropout layer, we randomly eliminate at every training step some link in order to avoid overfitting.

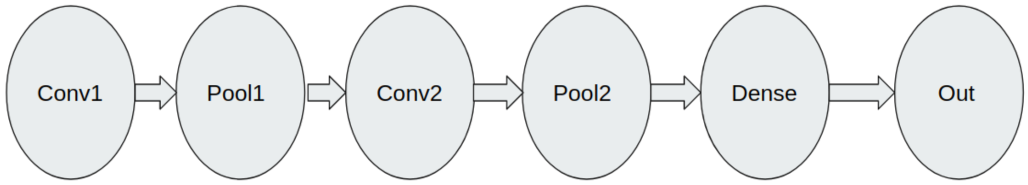

Figure 2: Topology of the our CNN.

5.2 Train and Test

After defining our model we have to train it with the collected data. Unfortunately with all the model of Neural Network above mentioned, we get results that do not satisfy our expectation. This can be explained from the fact that, until some point, the net saturates its capacity of learning, and perhaps doesn't improve again the result. The metric used to evaluate is the mean square error (**MSE**) calculated as the mean of the squared difference between the real wait time and the predicted wait time in seconds.

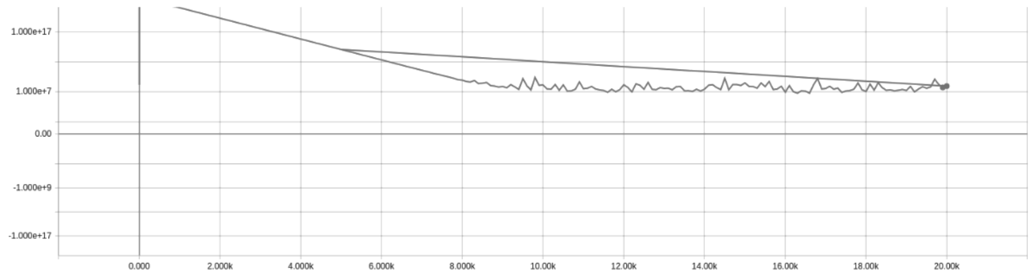

Figure 3: Trend of the error (MSE) while training and testing.

6 Boosted Decision Tree

This machine learning approach is based on ensemble of decision tree. A **decision tree** is a special tree where at each level one or more feature are used to choose the path that reach the leaf that contain the right result. A single tree is weak learner because it cannot represent a complex model. But if we combine a large number of trees with a weighted operation like sum, we can obtain a stronger learner that could properly predict our model. We use the *XGBoost* library to implement this model in our program.

6.1 Model

To implement our forest of trees we have to choose many parameters. First we set the properties of the single tree, like the max depth, the minimum child weight that is sum of weight needed in a child, the number of features used in each level and the number of sample used in each training instance.

We have also properties of the full ensemble like the total number of trees, the learning rate and the gamma number, that is the minimum loss reduction required to make a further partition on a leaf node of the tree. We tune this parameters doing some examples with reduced dataset and evaluating the results.

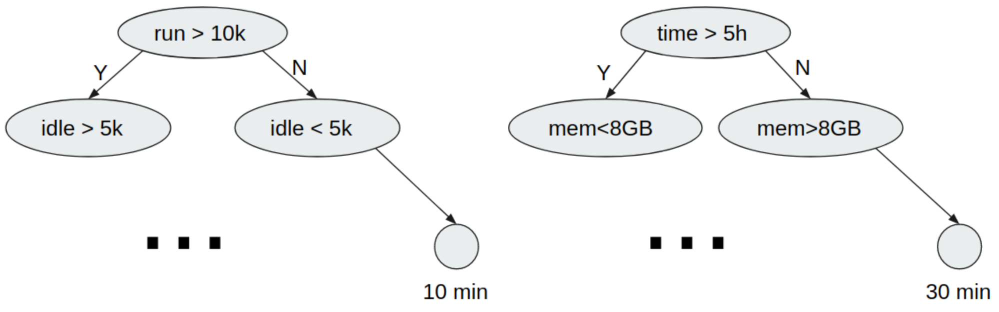

Figure 4: Example of decision trees that could be used to make prediction.

Finally to get better result, we preprocess the data and add some **Crossed** features, that are calculated combining many original features. They can highlight some type of knowledge that otherwise our model might not learn but is apparent from prior human knowledge. In our case we add a type of percentage usage within the experiment of which job is a member. We calculate these two percentage and add them to the input features.

$$Avaibility = \frac{quota - run}{quota}$$

$$Load = \frac{quota - idle}{quota}$$

6.2 Train and Test

After tuned the model, we train it with all the train data, and save the result into a file for future use. We use the data within the months from February to August (included), that compose about 6 M entry (1 GB of data). We split them randomly into two groups, the training set composed by 67% and the test set composed by 33% of the total data. The results achieved are very encouraging, because they are near the goal that we have set at the beginning.

7 Result

The best results are obtained with the Boosted Decision Tree model. We have a distribution error with these characteristics expressed in minute :

- Median : 1.57
- Mean : 22.03 (with sign : -0.25)
- Root Mean Square Error : 90.41

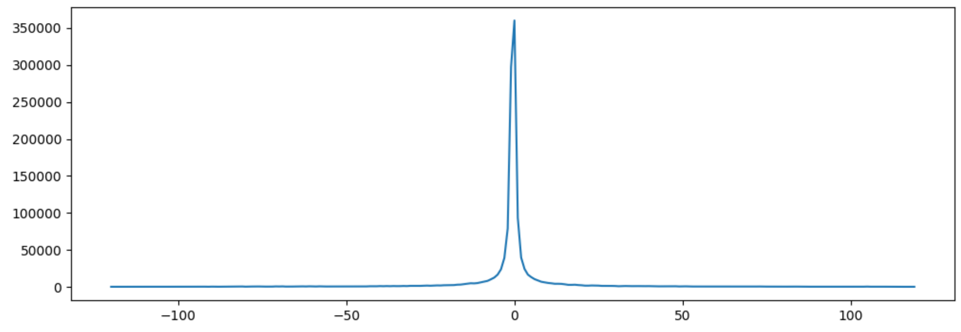

Figure 5: Error distribution over a **linear** scale. On y-axes we have the number of occurrences of the error valuated on x variable. On x-axes we have the error expressed in minutes. Negative one indicates an under estimation respect to the real time, positive one an over estimation.

If we retain acceptable an error lower than 15 minutes, in the 82 % of cases we have an hit. Otherwise if we expand this windows to 20 minutes, we can obtain until to the 85 % of hit.

If we focus on the error near the zero, we see that almost all (about the 90 %) the errors drop in the range of ± 45 minutes. In the other cases, we observe that the error is not symmetric with respect to the origin. Our model, indeed, tends to predict a value less than the real value, and this could potentially be a problem, because we create a false expectation in the user.

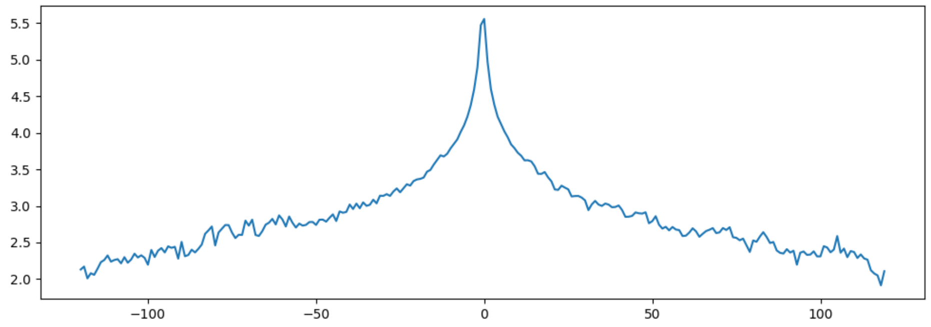

Figure 6: Error distribution over a **logarithmic** base 10 scale. On y-axes we have a logarithm number of occurrences of the error valuated on x variable. On x-axes we have the error expressed in minutes. Negative one indicates an under estimation respect to the real time, positive one an over estimation.

Abs error tolerated	N. of fails	Percentage of success
15 minutes	354,009	82.45 %
20 minutes	303,317	84.96 %
30 minutes	242,612	87.97 %
45 minutes	184,432	90.85 %
60 minutes	146,296	92.74 %
90 minutes	96,838	95.20 %

Table 1: Distribution of errors over many value of tolerating error.
Train on 4,096,690 sample. Test on 2,017,774 sample.

Furthermore we can ignore small errors on jobs with wait time over 4 hours because in a real scenario the user doesn't care that the precision of the value is too high. We also eliminate the sample and prediction that have both a value greater than 10 hours. With this premise, we increase the quality of our prediction to this characteristic (for 15 minutes) :

- Median : 1.24
- Mean : 16.57 (with sign : 0.079)
- Root Mean Square Error : 66.53

7.1 Evaluation

With the above consideration we can say that with a good margin of error, we get trusted results. Unfortunately this assert is true only if some assumption that we have done are verified.

When we have collected the data we have assumed that in a time frame of the last nine month, we get an homogeneous and valid pool of samples. This is only in part true because could happen event like the begin of a new experiment, big changes in total available resources, etc, that our model can't handle without a retraining. Moreover we assume that policies used to manage the queue will not change.

Ultimately, if we consider the actual situation, we can say that we are able to predict, with reasonably error, the time that a submitted job needs to start its execution.

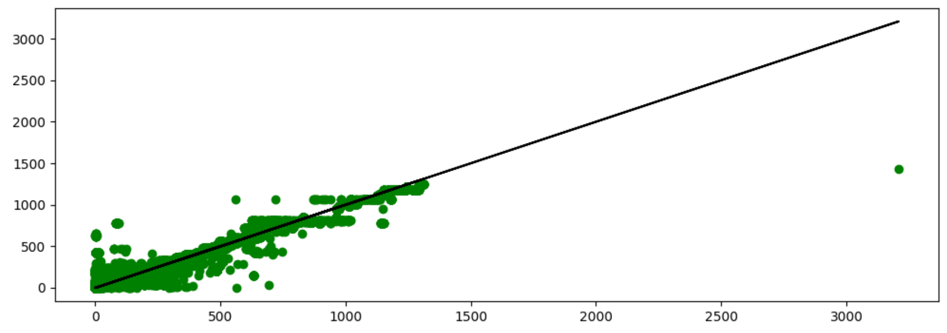

Figure 7: Distribution of the sample respect to the ideal prediction. On x-axes we have the real time, on the y-axes the predict one. Ideally the point should be the most possible near the black line.

7.2 Possible Improvements

In order to get better results we can do some easy improvement.

First of all, we can increase the number of training data used, adding other features like the priority of the user when submit the job, track other experiments and others parameter. This increases the knowledge of the system, and helps improve the model to better learn some additional rules. Then we can calculate other crossed features that explain high level concepts that otherwise the model might not consider.

We can also consider all the jobs that have been suspended (but not killed) for some reason, because even if they are not running, they possibly have an impact over the system queue.

Finally we can tune better some specific parameter and find the ones that obtain better result, we can increase the time of computation and explore more deeply some path early pruned in order to build a larger model.

8 Conclusion

In this project we have tried to predict the wait time of submitted job in the queue. This depends on the termination of executing jobs, and this is in literature a classically unpredictable problem. Therefore we have decided to have a compromise, accept approximate results, and adopt a machine learning model. After adopting various kinds of models like neural networks and boosted decision trees, we take the best results obtained and verifies if they satisfied our needs. We expect results with a negligible error respect to the user perceptions, and in almost all cases we obtain this. Unfortunately there are some special cases that our model barely handle because they represent uncommon situation. Moreover in some cases we have a wait time that exceed many hours, and then in these situation we are not interested to have a particular precision. Summarized we can affirm that we are able to predict the wait time with the condition above explained, in future we can improve this result, but at the moment the goal that we are preset is reached.